(4)

RADC-TR-88-190
Final Technical Report
September 1988

PERFORMANCE EVALUATION OF PARALLEL ALGORITHMS AND ARCHITECTURES IN CONCURRENT MULTIPROCESSOR SYSTEMS

Clarkson University

Sponsored by
Strategic Defense Initiative Office

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

APPROVED: [illegible]

ALAN N. WILLIAMS, 1Lt, USAF
Project Engineer

APPROVED: Raymond P. Urtz Jr

RAYMOND P. URTZ, JR.
Technical Director
Directorate of Command & Control

FOR THE COMMANDER: James W. Hyde III

JAMES W. HYDE III
Directorate of Plans & Programs

PERFORMANCE EVALUATION OF PARALLEL ALGORITHMS AND ARCHITECTURES IN CONCURRENT MULTIPROCESSOR SYSTEMS

H. H. Ammar
Y. S. Fong
R. Mukundan
C. A. Pomalaza-Raez

Contractor: Clarkson University
Contract Number: F30602-81-C-0169
Effective Date of Contract: 29 January 1987
Contract Expiration Date: 30 September 1987
Short Title of Work: *Performance Evaluation of Parallel Algorithms and Architectures in Concurrent Multiprocessor Systems*
Period of Work Covered: Jan 87 - Sep 87

Principal Investigator: Y. S. Fong
Phone: (315) 268-2126

RADC Project Engineer: Alan N. Williams, 1Lt, USAF
Phone: (315) 330-2925

This research was supported by the Strategic Defense Initiative Office of the Department of Defense and was monitored by Alan N. Williams, 1Lt, USAF, RADC (COTC), Griffiss AFB NY 13441-5700 under Contract F30602-81-C-0169.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS N/A
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) N/A	5. MONITORING ORGANIZATION REPORT NUMBER(S) RADC-TR-88-190
6a. NAME OF PERFORMING ORGANIZATION Clarkson University; 6b. OFFICE SYMBOL *(If applicable)*	7a. NAME OF MONITORING ORGANIZATION Rome Air Development Center (COTC)
6c. ADDRESS *(City, State, and ZIP Code)* Dept of Electrical & Computer Engineering Postsdam NY 13676	7b. ADDRESS *(City, State, and ZIP Code)* Griffiss AFB NY 13441-5700
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Strategic Defense Initiative Office (SDIO); 8b. OFFICE SYMBOL *(If applicable)*	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F30602-81-C-0169
8c. ADDRESS *(City, State, and ZIP Code)* Office of the Secretary of Defense Wash DC 20301-7100	10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO	WORK UNIT ACCESSION NO
63223C	B413	03	P2

11. TITLE *(Include Security Classification)*
PERFORMANCE EVALUATION OF PARALLEL ALGORITHMS AND ARCHITECTURES IN CONCURRENT MULTIPROCESSOR SYSTEMS

12. PERSONAL AUTHOR(S)
H.H. Ammar, Y.S. Fong, R. Mukundan, C.A. Pomalaza-Ráez

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT *(Year, Month, Day)*	15. PAGE COUNT
Final	FROM Jan 87 TO Sep 87	September 1988	26

16. SUPPLEMENTARY NOTATION
N/A

17. COSATI CODES			18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
FIELD	GROUP	SUB-GROUP	
12	07		Multiprocessing; Concurrent Processing; Processor Evaluation Tools; Trace-Driven Simulations

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*
This report describes the research effort directed towards the study of issues concerning concurrent systems. Specifically the report concentrates on the design, evaluation, and tuning of application programs on parallel architectures. Issues including mapping algorithms to architectures, and parallel programming support tools are also discussed. It is recognized that the currently available concurrent system analysis tools are not adequate in determining the detail performance of the application programs on specific architectures. To remedy this shortcoming, we recommend the development of a multiprocessor trace-driven simulator. This simulator will be beneficial to the evaluation on the performance of the SDI battle management algorithms on specific concurrent systems.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL Alan N. Williams, 1Lt, USAF	22b. TELEPHONE *(Include Area Code)* (315) 330-2925	22c. OFFICE SYMBOL RADC (COTC)

DD Form 1473, JUN 86 *Previous editions are obsolete.* SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

I. INTRODUCTION

The complexity in a concurrent system is formidable. There are multiple processors executing programs concurrently. The communications and synchronizations among individual or groups of processors are vital to the success of program execution. Resource contention and information flow requirements add their tolls to the overall load of the system. On top of this, the partitioning and balancing of the task in the parallel architecture put more stress on the overall system. It is essential that, under this tremendous complexity, a user has an indication of the performance of a certain task in such a system. In other words, the user has a knowledge of the utility and efficiency of the processors, can detect where the execution or communication bottleneck exists, and will be able to modify or fine-tune the program, the parallel architecture, or both to optimally execute the task.

This report describes the research effort carried out by this group in the general area of performance evaluation in concurrent systems. Specifically, the issues discussed include the mapping of algorithm onto parallel architecture, the parallel programming software tools, and the evaluation and tuning of algorithms on parallel architectures.

As alluded to in the next section multiprocessor systems are usually classified as loosely-coupled or tightly-coupled. The main concern of this report is the tightly-coupled system. But one of the issues discussed, namely the mapping of algorithms onto parallel architectures, is more of an issue in the loosely-coupled system. Thus the report starts with a discussion of issues in the loosely-coupled concurrent system, then it enters a short survey of the currently-available parallel programming tools on tightly-coupled systems. Finally the report presents an analysis of the performance evaluation issue and recommends the implementation of a multiprocessor trace-driven simulator. This simulator will be beneficial towards the analysis of the performance of SDI Battle Management Algorithms on specific concurrent processing systems.

II. MULTIPROCESSOR ISSUES

Multiprocessing systems can be divided into two main categories, loosely coupled, and tightly coupled systems. Loosely coupled systems have a distributed memory architecture and implement interprocess communication (IPC) using the message passing model. Examples of such systems are the intel's iPSC, the Ncube, the FPS T-series, and the Ametek's System 14. Tightly coupled systems, on the other hand, have for the most part a shared memory architecture and implement a shared memory and/or a message passing IPC model. Examples of such systems are the Encore Multimax, the Sequent Balance 8000, the Flex/32, the Alliant FX/8, and the BBN Butterfly. IPC delays are much more costly in loosely coupled systems than in tightly coupled ones. Measurements on the intel's iPSC hypercube have shown that the delay encountered in sending a message from a node to one of its nearest neighbors is 1 ms which is quit excessive for applications with frequent *communications* and *synchronizations*.

It is well known from the experiences gained in programming the above systems that shared memory systems are much easier to program than distributed memory systems. This is due to the mapping problem that exists in distributed memory systems which is discussed in more details below. However, debugging programs on a shared memory architecture is much more difficult than on a distributed memory system because in the later a certain task is assigned to a certain node in the system, therefore problems can be easily located.

In the next section a brief discussion of some of the features of loosely coupled multiprocessors will be presented. Emphasis is given to the problems of mapping and scheduling.

III. MAPPING AND SCHEDULING ISSUES

Suppose that for the given computational task a suitable algorithm has been provided. Assume that this algorithm has been divided into some number of subtasks in such way that some of these subtasks can be run concurrently. As these subtasks run (on the different processors), they may need to exchange data amongst themselves, and it may not be possible to start the execution of one subtask before the completion of some other subtasks. In order to estimate the behavior of the algorithm on the system it is necessary to estimate the extent of this data communication between subtasks and their data dependence. The *scheduling* problem is then to find the optimum allotment of these subtasks among the processors so that the maximum possible speedups (or any other performance parameter) is achieved. For example subtasks that exchange data frequently should be allotted to processors that are adjacent, i.e., directly connected to each other.

It can then be said that having several subtasks and their relationships with respect to data dependency, and several processors (not necessarily identical), with their interconnection scheme one needs to *map* one on the other. In a most ideal system little information will be available beforehand on the nature of the tasks being performed and this mapping problem has to be done at runtime, dynamically, as subtasks are created. The time spent on this mapping will add to the total time that it takes to complete the task. This mapping time cannot in general be ignored and will degrade the performance of the system, specially in real-time applications, where time is of the greatest importance. A lot of work has been done in regarding the problem of allocating *noninteracting* tasks in a distributed environment. Not too much attention has been paid to the data dependency of the communication amongst the tasks. However in a multiprocessor environment, interprocessor communication overhead play an important part in determining the performance of the system.

In [1] a graph model of a modular program and the maximal-flow algorithm is used to obtain an optimal assignment of subtasks or modules to a *two-processor* system. In the graph model each module is represented by a node and the weight on the edge connecting two nodes represents the costs of an intermodule reference when the two nodes are assigned to different computers. Each cutset of this graph partitions the graph into two disjoint subsets, and each subset could be assigned to each processor such that the weight of the edges comprising the cutset could be minimized. This will translate in a minimization of the total runtime. Generalizations for more than two processors can also be made.

A more general case of this mapping problem is described in [2]. It is assumed there that most of the multiprocessors systems are incompletely connected, that is, a direct link does not connect each pair of processors. When assigning modules to processors, pairs of modules that communicate with each other should be placed, as far as possible, on processors that are directly connected. It is first shown that this problem is very difficult and that an optimum solution can be seldom found in a practical manner. It then describes an heuristic algorithm that was developed to solve this problem for a specific array processor.

Another method called the *wave scheduling* is proposed in [3]. Here it is assumed that there is a hierarchical level of control in the form a tree and that it is wanted to execute many independent parallel programs *simultaneously* in a large multiprocessor. The collection of tasks that constitute a parallel program is called a task force. A task force needing a specific number of nodes is entered in a queue at any node, and the hierarchical control at the root of the subtree schedules the enqued task force that are no larger than the number of nodes in the subtree. This *decentralized* wave scheduling works well with low to moderate works loads and its efficiency is comparable to that of a centralized scheduling.

For the case where neither all the processors nor all the channels bandwidths are identical [30] describes a procedure that again is based in gra ˙ ;heoretic results. The algorithm proposed makes use of two graphs, the *Computational Flow Graph*, (CFG), and the *Computation Resource Graph*, (CRG). The CFG is a directed graph that shows the data dependency or the interrelationships with respect to data exchange between the various subtasks. Each node represents a subtask and each edge has a value that represents the amount of data that will need to be transferred. The CRG is a undirected graph in which the nodes are the processors and the edges denote the channel between the two adjacent processors. The mapping algorithm accepts these two graphs as input, and produces as output the mapping of the CFG on the CRG. Once the mapping is done, a simulation program is run that will give an estimate of some important parameters that can be used to evaluate the performance of the architecture/algorithm combination that has been employed.

In all of these methods it is assumed that the algorithms to be used have already partitioned into their respective subtasks. This is still being done mostly by the programmer. Work is currently underway to develop intelligent compilers that can extract parallelism from programs written for sequential machines.

IV. PARALLEL PROGRAMMING SOFTWARE TOOLS

This is a short survey section that intends to summarize available software tools for parallel programming currently on the market. By software tools for parallel programming we mean the following three categories of software aids:

1. Parallel program performance evaluation software and simulator
2. Parallel program debugger
3. Other parallel programming support.

This survey is very brief, and does not intend to be complete. It merely provides a random sampling of the current market situation. The information is taken from the documents provided by the vendors.

Performance evaluation tool.

Sequent Balance series

The DYNIX operating system of the Sequent Balance series machine provides several utilities which can be used to measure the performance of the program. These utilities are:

1. **prof** – This command reports the number of times each subroutine is called, and the amount of time spent in each subroutine.
2. **gprof** – This command is similar to **prof**, but also prints a call graph that shows profiling data in the context of the program's subroutine call hierarchy.
3. **size** – This command reports the amount of memory used by a program's machine code and data structure.
4. **time** – This command reports the program's totai execution time, in terms of both real time and CPU time. Other options are also available to report more information about the program being timed, such as memory size, page faults, I/O operations, and context switches. In parallel situation, the report can be prepared for each process, not just for the whole application.

Alliant FX/Series

Concentrix, the Alliant operating system, is a full native port of the Berkeley Unix operating system and includes programming and system management tools required for an efficient use of the architecture. There is a system monitor (**mon**) that graphically details the percent utilization of all system resources. This tool enables the "tuning" of a multiuser system to provide balanced support for large and small production jobs. **mon** monitors the utilization of system resources and display status information in real time. It has three modes of operation: family mode, system mode, and computing resource mode. Thus for example in the family mode **mon** monitors a target process and all its descendent process.

At the programming level there are facilities such as **gprof** that collects and reports timing information on a per-routine basis.

Debugging tool.

Sequent Balance series

The DYNIX system provides the **Pdbx** debugger to enable a programmer to debug a compiled C, FORTRAN, or Pascal program without having to understand the assembly language output produced by the compiler. **Pdbx** is a version of the UNIX 4.2bsd **dbx** debugger that has been enhanced to support the debugging of programs that consist of multiple concurrent processes. **Pdbx** allows the programmer to set break points on processes, stop one or more processes based on events in a specified process, or step through a program. **Pdbx** also supports the debugging of applications that consist of multiple processes created from different programs.

Alliant FX/Series

Concentrix also provides a version of the **dbx** debugger. Its capabilities are very similar to the one from the Sequent series.

Parallel programming support.

Sequent Balance series

The Sequent DYNIX operating system also contains other supports for parallel programming.

1. Parallel programming library - The DYNIX Parallel Programming Library simplifies the use of shared memory and supports the most commonly used parallel programming mechanisms. This library is designed for use with homogeneous multitasking applications written in C, FORTRAN, or Pascal.
2. FORTRAN Parallel Processor - In FORTRAN programs, most loops are implemented as `DO` loops, so Sequent provides a parallel processor that automatically converts

selected FORTRAN DO loops to execute in parallel.

Alliant FX/Series

The Fortran compiler has options that automatically or with very little human intervention provide concurrency and vectorization optimizations. These are in addition to the general and directive optimizations options. At the present time only, the Alliant FX/Fortran and the compiler developed for the Warp processor are the only compilers that have these capabilities.

Occam and the Inmos Transputer

Occam is a computer language for parallel programming. It is based upon the concept of parallel execution. It also provides automatic communication and synchronization between concurrent processors. The Inmos Transputer is a single-chip computer whose architecture facilitates the construction of parallel processing systems. The Transputer executes occam programs more or less directly. It can be say that occam is the assembly language of the Transputer.

V. DESIGN, EVALUATION, AND TUNING OF APPLICATION PROGRAMS ON PARALLEL ARCHITECTURES

Performance is one of the key issues that needs to be taken into account in the design, development, configuration and tuning of parallel algorithms and architectures. The evaluation of a system also include such factors as ease of use, availability, reliability, and security. Achieving maximum performance involves not only tuning the parallel algorithm for a given architecture (e.g., adjusting the granularity of parallelism and the distribution of data between the shared and local memories) but also tuning the underlying architecture, since often minor changes in the architecture can cause a significant improvement in performance.

In general computer performance evaluation methods are divided into three main areas (see [4] for a good survey), namely performance measurements, analytic performance modeling, and simulation performance modeling. Performance measurements [10] is of course suitable only for tuning the parallel algorithm on an existing running architecture. However modeling is required in order to otherwise predict the performance under different architectural changes. One of the principle benefits of performance modeling, in addition to the quantitative prediction obtained, is the insight into the structure and behavior of the system that is obtained by developing a model. However, such models must first be validated using performance measurements on an existing architecture before they can be used with greater confidence to investigate the performance effects of design enhancements and configuration changes.

Performance models span the range from simple analytically tractable models to very detailed trace-driven simulation models. Analytical models for parallel processing were discussed in [5-9]. However, such models can't be easily extended to incorporate all the delays encountered during the execution of a parallel program on a parallel architecture. For example the model proposed in [9] does not consider delays due to contention for shared memory or overhead delays due to task scheduling and synchronization. This is of course due to the complexity of parallel processing systems, therefore a detailed analytic model will rapidly become intractable [6,11]. The development of simple tractable and accurate analytical models for parallel processing systems is still an active area of research.

Trace-driven simulation models, though more expensive to develop and less flexible to change than analytic models, allow a detailed and precise modeling of the various activities in the system. And by utilizing parallel simulation techniques [12,28], the execution of an entire application program amounting to hundreds of millions of instructions on a parallel architecture can be efficiently simulated. Moreover using modular design techniques a simulation model can be developed to simulate the execution of a parallel application program on several parallel architectures.

A methodology for tuning algorithms to architectures is described below. A simple implementation of a trace-driven simulator of a shared memory multiprocessor system which demonstrates the performance tuning methodology is described in [29].

Performance evaluation methodology.

In this section, using the above mentioned remarks, we describe a methodology for evaluating the performance of an application algorithm on a parallel architecture. Fig. 1 summarizes the various steps needed as well as the supporting tools. The user interacts with a parallel program development tool with which he could define and code a parallel program for his application. A trace-driven simulator is then used to simulate the execution of the parallel program on predefined parallel architecture. The simulator produces a trace file and a summary of some performance statistics users would like to know right after the simulation run (e.g., the speedup and the utilization of each CPU). The trace file generated can be used to obtain a more detailed analysis of the various performance bottlenecks in the system. This is accomplished by a trace analysis tool equipped with graphics capabilities to aid the user in analyzing different parts of the system. The user can then go back and tune the program or the architecture to eliminate bottlenecks. In the rest of this section we discuss in more details each of the above mentioned tools.

Parallel Program Development Tools.

Tools for parallel program development can range from tools which provide the programmer with locks and synchronization primitives to tools which provide automatic parallelization of sequential code [21,22]. The primitives can be in the form of library of subroutines or language extensions see for example [14-18]. They leave all the problems of program partitioning, scheduling, and synchronization for the programmer. While some experience have already been gained in developing parallel programs using such tools [19,20], the process can prove to be tedious for large application programs. However, at the other extreme, the later tools which does not leave to the programmer any control over parallelism are not yet efficient enough since the only automatic tool available to date is limited to individual loops.

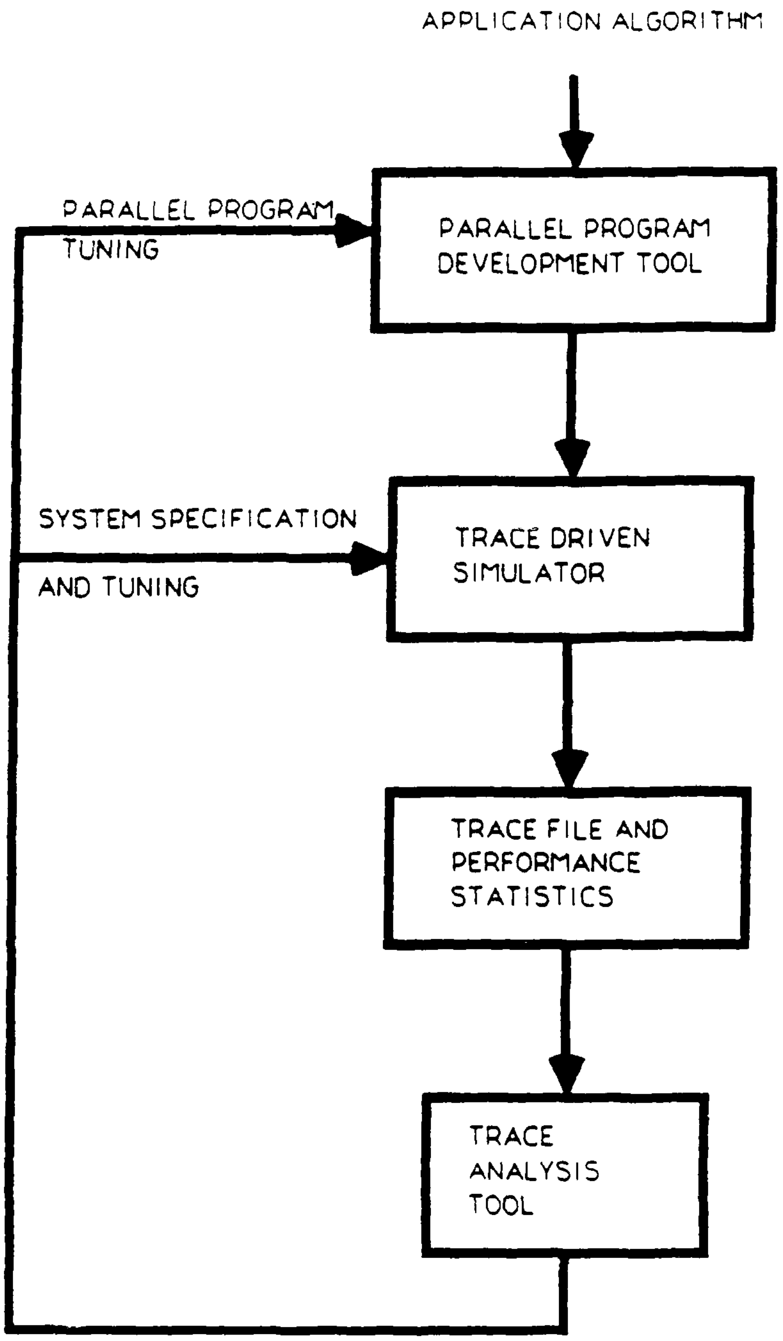
APPLICATION ALGORITHM
PARALLEL PROGRAM TUNING
PARALLEL PROGRAM DEVELOPMENT TOOL
SYSTEM SPECIFICATION AND TUNING
TRACE DRIVEN SIMULATOR
TRACE FILE AND PERFORMANCE STATISTICS
TRACE ANALYSIS TOOL

Fig. 1 Performance Evaluation Methodology

In [24] a computer-aided multiprocessor programming tool (CAMP), which follows an approach in the middle of the above two extremes, is described. CAMP let the user defines or extracts different program segments. It then helps in partitioning these program segments into a number of processes, and inserting synchronization primitives where needed. The user is involved in either decision making (which segment is to be parallelized and to what extent) or the writing of program codes in parallel format possibly after being restructured by CAMP. For example the user may extract an iterative segment (a loop) of the algorithm he trying to code and if the iterations are independent he can write it as a parallel loop. However, if the iterations are dependent CAMP will attempt loop alignment or restructuring techniques [24] to eliminate or minimize synchronization between iterations.

The FX/Fortran compiler developed for the Alliant multiprocessor [23] implements some of the ideas mentioned in CAMP. It permits the user to control parallelism at the program, subprogram, and loop levels. When enabled by the user, the compiler optimizes program segments for concurrency and vectorization. It analyzes programs for data dependencies and generates an optimized executable code for DO loops and DO WHILE loops.

The above tools are not only important for parallel programs development, but also for the tuning of such programs to the underlying parallel architecture.

Simulation tools.

Although the program development tool CAMP described in the previous section incorporates a simulator which estimates the speedup achieved for different partitioning and synchronization strategies. It does not provide a detailed trace and performance statistics for the execution of a program. Simulation tools which rely on a simplied model for the system often miss some of the important factors that may greatly influence system performance. For example in a shared memory multiprocessor with a multistage interconnection network (IN), a simplified simulation model might assume a fixed average value for the delay encountered in the IN for each shared memory access. If hot spots [25] occur often during the execution of a program such a delay will have a tremendous variability which have a significant impact on performance.

Stochastic simulation models, though can be made quite general and easily adaptable to changes, are faced with the problem of estimating the *distributions of the random* variables which define the model (e.g., the time between two consecutive shared memory accesses).

As mentioned earlier, trace-driven simulation models which simulate the execution of a program statement by statement on a given architecture are the most precise. However, they are also the most expensive in terms of computer time. In [28], PSIMUL, a trace-driven simulator developed for the study of memory access patterns of parallel programs, is described. The information generated from PSIMUL is mainly about memory reference patterns of each simulated process, therefore, observations on shared and private data, their frequency of access and locality can be obtained. It is developed for the RP3 architecture [26] and for application programs written using the VM/EPEX parallel processing environment [17].

In [27], MULTIMOD, a trace-driven simulation model for a multiprocessor, is described. The architecture modeled is a shared memory system with a Banyan or a full crossbar interconnection network for data and synchronization and a ring network for task handling. The model accepts an ordinary Fortran source program and augments it with special functions to support distributed task creation and synchronization. Such functions are used to parallelize DO loops only without user control. The model is also not amenable to architectural variations as proclaimed except for the interconnection network to some extent, and is not suitable for simulating large application programs.

A parallel trace-driven simulator model can be developed to efficiently simulate large application programs. Such a model is inherently parallel since it simulates a parallel processing system. Therefore, it can be developed and executed efficiently in a parallel processing environment such as the Alliant system. Moreover, using modular design techniques, the model can be made amenable to architectural variations to improve performance and to accommodate new ideas.

Trace Analysis Tools.

A trace analysis tool is a tool that analyzes the trace file and produces detailed accounts of resource usage, data access behavior, and the various overhead delays encountered during the execution of the parallel program. There can be many passes through the trace file, each producing different reports or displays. Such tools are usually designed for interactive use and equipped with graphics capabilities.

In [10], Monit, a performance monitoring tool for parallel programs, is described. Monit uses trace files that are generated during the actual execution of parallel programs written in the parallel processing environment PPL on a Sequent Balance 8000 multiprocessor system. Therefore, the trace files produced contains slightly distorted results due to the data collection overhead during actual program execution on the Sequent. Monit analyzes these trace files and produces time oriented graphs of resource usage and system queues. Users interactively select the displayed items, resolution, and time intervals of interest. The current implementation of Monit is for the SUN-3 workstation. The program however is easily adaptable to other systems and efforts to generalize its use to analyze traces produced by simulators are underway.

VI. CONCLUSION

The major issue presented by this report is the performance evaluation of *large* parallel programs on *large* parallel architecture. It is an important issue in concurrent systems, specially more so in connection with the SDI Battle Management requirements. From the discussion in the report, it is obvious that currently there is no readily-available good performance evaluation tools. To adequately simulate and analyze the performance of large parallel programs on concurrent system, such as the case in SDI Battle Management Algorithms, a parallel trace-driven simulator must be developed. This simulator will be able to simulate accurately and efficiently the large application programs. Moreover, the modular-type construction will enable the modification of the simulator due to architectural changes and to accommodate new proposals.

We thus recommend the development of a parallel trace-driven simulator for the evaluation of parallel programs on concurrent systems. A preliminary version of the simulator has been developed [29]. This simulator can be generalized to accommodate the requirements of evaluating larege application programs. Some analytic modeling of subsystems needs to be investigated, and the simulator needs to be validated with actual system measurements. We believe the development of this simulator will benefit the goal of evaluating the SDI Battle Mangement Algorithms on various concurrent systems.

REFERENCES

[1] Stone, H. S., "Multiprocessor Scheduling with the Aid of Network Flow Algorithms", *IEEE Trans. Software Engineering*, vol. SE-3, pp. 85-94, January 1977.

[2] Bokhari, S. H., "On the Mapping Problem," *IEEE Trans. Computers*, vol. C-30, No. 3, pp. 207-214, March 1981.

[3] Tilborg, A. M. and Wittie, L. D., "Wave Scheduling-Decentralized Scheduling of Task Forces in Multicomputers," *IEEE Trans. Computers*, vol. C-33, No. 9, pp. 835-844, September 1984.

[4] Heidelberger, P., and S. Lavenberg, "Computer Performance Evaluation Methodology," *IEEE Trans. on Comp.*, vol. C-33, Dec. 84.

[5] Ammar, H., and R. Liu, "Hierarchical Models for Parallel Processing Systems Using the Generalized Stochastic Petri Nets," *Proc. 1984 ICPP, IEEE Comp. Soc.*, Aug. 1984.

[6] Ammar, H., "Analytical Models for Parallel Processing Systems", Ph.D. Dissertation,

Elec. and Comp. Engr. Dept., University of Notre Dame, 1985.

[7] Thomasian, A., and P. Bay, "Analytic Queuing Network Models for Parallel Processing of Task Systems", *IEEE Trans. Comp.*, vol. C-35, Dec. 86.

[8] Cvetanovic, Z., "The Effect of Problem Partitioning, Allocation and Granularity on the Performance of Multiple Processor Systems",*IEEE Trans. Comp.*, vol. C-36, April 87.

[9] Nelson, R., D. Towsley, and A. N. Tantawi, "Performance Analysis of Parallel Processing Systems",*Proc. 1987 ACM SIGMETRICS Conf.*, Banff, Alberta, 1987.

[10] Kerola, T., H. Schwetman, "Monit: A Performance Monitoring Tool for Parallel and Pseudo-Parallel Programs",*Proc. 1987 ACM SIGMETRICS Conf.*, pp163-174.

[11] Marsan, M. A., et al, "Modeling the Software Architecture of A Prototype Parallel Machine" *Proc. 1987 ACM SIGMETRICS Conf.*, pp175-185.

[12] Reed, D. A., et al, "Parallel Discrete Event Simulation: A Shared Memory Approach", *Proc. 1987 ACM SIGMETRICS Conf.*, pp36-38.

[13] Darema-Rogers, F., et al, "Memory Access Patterns of Parallel Scientific Programs", *Proc. 1987 ACM SIGMETRICS Conf.*, pp46-58.

[14] Jordan, H. F., *Structuring Parallel Algorithms in an MIMD Shared Memory Environment*, Report no. CSDG84-2, Dept. of Elect. and Comp. Engr., University of Colorado.

[15] Mehrotra, P., and J. Van Rosendale, *The BLAZE Language: A Parallel Language for Scientific Programming*, ICASE Report no. 85-29, NASA Langley Research Center, Langley, VA 1985.

[16] Pratt, T. W., *PISCES: an Environment for Parallel Scientific Computations*, ICASE Report no. 85-12, NASA Langley Research Center, Langley, VA 1985.

[17] Darema-Rogers, F. et al, *An Environment for Parallel Execution*, IBM Research Report #11225, Yorktown Heights, NY 1985.

[18] Lusk, E. L., and R. A. Overbeek, *Implementation of Monitors with Macros: A Programming Aid for the HEP and Other Parallel processors,* Report no. ANL-83-97, Argonne National Laboratory, Argonne, Ill. 1983.

[19] Davidson, G. S. *A Practical Paradigm for Parallel Processing Problems*, Report no. SAND85-2389, Sandia National Laboratories, Albuquerque, NM, 1986

[20] Karp, A. H. "Programming for Parallelism," *IEEE Comp.*, *IEEE Comp. Soc.*, May 1987.

[21] Kuck, D., "High Speed Multiprocessing and Compilation Techniques," *IEEE Trans. Comp.*, C-29, 1980, pp763-776.

[22] Allen, J. R., and K. Kennedy, " PFC: A Program to convert Fortran to Parallel Form," *in Supercomputers: Design and Applications,* K. Hwang (ed.), IEEE Comp. Soc., 1984.

[23] Perron , R., and C. Mundie, "The Architecture of the Alliant FX/8 Computer," *Digest of Papers, Compcon*, Spring 86, IEEE Comp. Soc., 1986.

[24] Peir, Jih-kwon, and D. Gajski, "CAMP: A Programming Aide For Multiprocessors," *Proc. 1986 ICPP, IEEE Comp. Soc.*, pp475-482.

[25] Pfister, G. F., and V. A. Norton, "Hot Spot Contention and Combining in Multistage Interconnection Networks," *IEEE Trans. Comp.*, C-34, Oct. 1985.

[26] Pfister, G. F. et al, "The IBM Research Parallel Processor Prototype (RP3): Introduction and Architecture," *Proc. 1985 ICPP, IEEE Comp. Soc.*, pp764-771.

[27] Wirsching, J. E., *A Multiprocessor Model With Distributed Task Creation*, Report no. UCRL-53648, Lawrence Livermore National Laboratories, Livermore, CA, 1985.

[28] So, K. et al, *PSIMUL- A System For Parallel Simulation of the Execution of Parallel Programs*, IBM Research Report #RC11674, 1986.

[29] Ammar, H. H., *On the Design, Evaluation, and Tuning of Application Programs on Parallel Architectures*, Technical Report, Department of Electrical and Computer Engineering, Clarkson University, Aug 1987.

[30] Agrawal, D.P, et al, "Evaluating the Performance of Multicomputer Configurations", *COMPUTER*, pp. 23-37, May 1986.